Universidad Politécnica de Madrid
Facultad de Informática

Trabajo de Fin de Grado

Implementación de una herramienta de productividad para el desarrollo y despliegue de aplicaciones

Autor: Sonia Martínez Martín

Tutor: Genoveva López Gómez

Madrid, 6 de junio de 2014

Agradecimientos

Después de años de sufrimiento por fin hemos llegado a la línea de meta, y no será que no ha costado, porque... tela. Antes de entrar en harina y ponernos técnicos me gustaría emplear unas líneas para dar protagonismo a todas esas personas que me han hecho llegar hasta aquí, porque yo sola, probablemente, no habría pasado de segundo (de tercero sí, pero segundo va antes, y ni de broma). De ahí ese plural de la primera frase de esta página, porque considero que este camino no lo he recorrido sola prácticamente nunca. De una forma u otra siempre he podido contar con alguien que me ayudaba y apoyaba ante las dificultades.

En primer lugar no puedo más que citar a mis padres, que no sólo me pagaban las matrículas cada semestre, a veces con precios desorbitados, sino que nunca han dudado que acabaría llegando aquí ni me han juzgado cuando no sacaba la nota esperada. Del mismo modo es imposible que me olvide de la Aya, esa segunda madre que nunca ha dejado de presumir de su nieta la informática y siempre me ha preguntado con tanto amor qué tal me iba (y no detallo más, porque si empiezo a hablar de su sopa, no acabo). Aunque la veo y hablo con ella mucho menos, también la Abuelita, mi otra abuela, se merece una mención, porque por cada examen que se ha enterado que tenía, una vela que me ponía y una oración (mínimo) que me dedicaba. Siguiendo con la familia, mis hermanos también deben aparecer aquí; Patricia porque lleva años haciendo los recados que podría haber hecho yo de no tener esos horarios interminables; y Raúl ser siempre tan sumamente atento conmigo cuando quiere algo, preguntándome si estoy muy ocupada y llevándose un sí por respuesta una vez tras otra... angelito. Acabo la sección familiar con una mención especial al Abu, porque aunque de mis años de universidad no vio más que unos meses, estoy segura de que estaría muy orgulloso de "su zapatera", y sé que siempre va conmigo, en cada caminata que me doy, que no son pocas.

En la Facultad me gustaría agradecer al profesorado haberme enseñado tanto, de una manera u otra, en especial a Javier Soriano y Genoveva López por haberme dado la oportunidad de trabajar en el laboratorio, donde en mucho menos tiempo, he aprendido tantísimo más. Y no sólo eso, sino que gracias a este laboratorio, el OIL, hemos formado ese variopinto y divertido grupo con el

que tanto hemos aprendido todos sin jamás bajar el tono de voz ni desperdiciar ni un momento para echar una carcajada. Agradezco así a Alberto (el Hambriento), Salva (el Autista), Fer (el Empresario) y David (el Moderno) el haberme acompañado estos últimos años de mi paso por la FI

Para desahogarme fuera de la Facultad siempre he podido contar con el baile, que tan importante es también para mí, y sin el cual estoy más que segura que jamás habría podido llegar hasta aquí; y dentro del baile, agradecerles todo a mis amigas y compañeras porque son únicas y con ellas es imposible encontrarse mal aunque estés hasta el cuello (Marie y Elisa por un lado; Laura, Bea y Sandra por otro; y, sobre todo, Almudena, compañera de batallas desde hace tanto tiempo). Por supuesto también a mi profesora y jefa ocasional cuando lo he necesitado, Inma Sáenz, que no contenta con enseñarme a bailar y alegrarme con su forma de ser en sus clases, me da trabajo como informática en sus proyectos, complementando mi formación.

Para el final he dejado lo mejor, además porque sé que si empiezo con él no dejo sitio para nadie más (y no sólo porque tenga el nombre más largo de toda esta sección). Gracias, gracias y gracias al señor Carlos David Rodríguez Peña (Carlos Daviiiiiiiiiiiiiiiis), ese chico tan alto con nombre de telenovela (o de disco compacto) que nada más empezar la carrera se convirtió en la persona más importante de mi vida y que desde entonces jamás ha dejado de estar ahí para cualquier cosa, ayuda, apoyo, aguantándome siempre, sin excepción (y mira que es difícil, ya lo dice mi madre). Es la persona sin la cual estoy segurísima no habría conseguido llegar hasta aquí, así que, una vez más, gracias por nunca dudar que pueda llegar a hacer algo, incluso cuando yo no lo veo ni de lejos. Por esos "sé que puedes" y esos empujones para conseguirlo cuando son necesarios.

A todos, de verdad, mil gracias.

Resumen

En la actualidad se está viviendo el auge del *Cloud Computing* (Computación en la Nube) y cada vez son más las empresas importantes en el sector de las Tecnologías de la Información que apuestan con fuerza por estos servicios. Por un lado, algunas ofrecen servicios, como Amazon y su sistema IaaS (Infrastructure as a Service) Amazon Web Services (AWS); por otro, algunas los utilizan, como ocurre en el caso de este proyecto, en el que Telefonica I+D hace uso de los servicios proporcionados por AWS para sus proyectos.

Debido a este crecimiento en el uso de las aplicaciones distribuidas es importante tener en cuenta el papel que desempeñan los desarrolladores y administradores de sistemas que han de trabajar y mantener todas las máquinas remotas de uno o varios proyectos desde una única máquina local. El ayudar a realizar estas tareas de la forma más cómoda y automática posible es el objetivo principal de este proyecto.

En concreto, el objetivo de este proyecto es el diseño y la implementación de una solución software que ayude a la productividad en el desarrollo y despliegue de aplicaciones en un conjunto de máquinas remotas desde una única máquina local, teniendo como base una prueba de concepto realizada anteriormente que prueba las funcionalidades más básicas de las librerías utilizadas para el desarrollo de la herramienta.

A lo largo de este proyecto se han estudiado las diferentes alternativas que se encuentran en el mercado que ofrecen al menos parte de la solución a los problemas abordados, pese a que los requisitos de la empresa indicaban que la herramienta debía implementarse de forma completa. Se estudió a fondo después la prueba de concepto de la que se partía para, con los conocimientos adquiridos sobre el tema, mejorarla cumpliendo los objetivos marcados.

Tras el desarrollo y la implementación completa de la herramienta se proponen posibles caminos a seguir en el futuro.

Abstract

Nowadays we are experiencing the rise of Cloud Computing and every day more and more important IT companies are betting hard for this kind of services. On one hand, some of these companies offer services such as Amazon IaaS (Infrastructure as a Service) system Amazon Web Services (AWS); on the other hand, some of them use these services, as in the case of this project, in which Telefonica I+D uses the services provided by AWS in their projects.

Due this growth in the use of distributed applications it is important to consider the developers and system administrators' roles, who have to work and do the maintenance of all the remote machines from one or several projects from a single local machine. The main goal of this project is to help with these tasks making them as comfortable and automatically as possible.

Specifically, the goal of this project is the design and implementation of a software solution that helps to achieve a better productivity in the development of applications on a set of remote machines from a single local machine, based on a proof of concept developed before, in which the basic functionality of the libraries used in this tool were tested.

Throughout this project the different alternatives on the market that offer at least part of the solution to the problem addressed have been studied, although according to the requirements of the company, the tool should be implemented from scratch. After that, the basic proof of concept was thoroughly studied and improved with the knowledge acquired on the subject, fulfilling the marked goals.

Once the development and full implementation of the tool is done, some ways of improvement for the future are suggested.

Índice de contenidos

Índice de figuras

Capítulo 1

Introducción

Los sistemas distribuidos siempre han sido uno de los mayores desafíos del software. El auge de la era tecnológica y el uso de internet han proporcionado los medios necesarios para que estos sistemas sean comunes a día de hoy, ofreciendo servicios hace no muchos años impensables. Más concretamente, el uso del *Cloud Computing* ha aumentado en los últimos años de forma exponencial. Ya sólo la búsqueda del término en Google demuestra el aumento de su popularidad con respecto a otro método de computación distribuida, el *Grid Computing*.

Figura 1. Evolución de la búsqueda en Google de los términos Cloud y Grid Computing. **Fuente**: Google, 2014

Como se puede comprobar en el gráfico de Google Trends (*Figura 1*) fue a principios de 2007 cuando la computación en la nube comenzó su popularidad, teniendo ésta su punto culmen hasta el momento en marzo de este mismo año, en el que superaba en números de 100 a 1 a las búsquedas con referencia a la computación grid.

En parte por esta popularidad, el uso de aplicaciones distribuidas ha ido creciendo en importancia, utilizándose cada vez más máquinas virtuales en lugar

de físicas para la ejecución de estas aplicaciones, ocasión que grandes empresas como Google, Amazon o Microsoft no han dejado escapar, ofreciendo desde hace años servicios hardware y software en la nube. Un ejemplo de esto es el presentado en la *Figura 2* [1].

Estos servicios se suelen clasificar en *Infrastructure as a Service* (desde ahora, IaaS), *Platform as a Service* (desde ahora Paas), y *Software as a Service* (desde ahora, SaaS) y sus características principales son el ser desplegados, configurados, gestionados y accedidos de forma programática y remota, el pago por uso (*pay-per-use*) por parte de los clientes, su alta escalabilidad, su elasticidad y su alta disponibilidad.

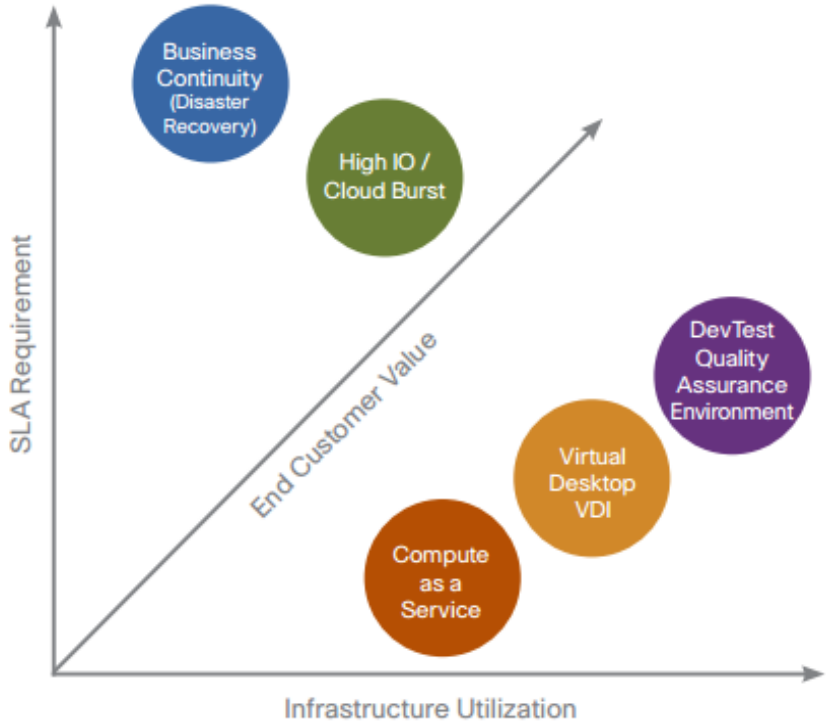

Figura 2. Ejemplo de ofertas IaaS de Cisco en función de varios factores. **Fuente:** Cisco Systems, Inc. 2009

Este Trabajo de Fin de Grado (TFG) se ha realizado en el marco de las actividades que se desarrollan en el Laboratorio de Innovación Abierta UPM – Telefónica I+D y consiste en el diseño y desarrollo de una solución software con capacidad para el desarrollo y despliegue de aplicaciones distribuidas en un conjunto de máquinas remotas desde una única máquina local.

De forma tradicional, el despliegue de aplicaciones se realiza por parte del desarrollador de forma manual, creando él mismo todas las máquinas virtuales

necesarias y poniéndolas a punto para ser utilizadas. Del mismo modo, no es habitual que la generación de infraestructura necesaria para un proyecto se haga en tiempo real. Con la herramienta resultante de este trabajo de fin de carrera (en adelante TFG) se pretende automatizar lo máximo posible estas tareas y facilitar su ejecución para una mayor rapidez por parte de todos los roles envueltos en un determinado proyecto.

El uso de la herramienta desarrollada irá enfocado al desarrollo de proyectos de cualquier dimensión, en principio pensando en el marco de empresas en las que un equipo trabaja de forma distribuida sobre el mismo proyecto desempeñando cada miembro diferentes papeles en el desarrollo. Gracias a esta herramienta se mejorará la productividad durante todo el ciclo de vida del proyecto sobre el que se use, agilizando su producción y facilitando la compenetración de las personas involucradas en el mismo.

El uso de esta herramienta se complementará con otra que gestiona las posibles configuraciones de las máquinas presentes en la red distribuida y que ha sido desarrollada en el TFG de un compañero del mismo laboratorio.

1.1. Motivaciones

La principal motivación a la hora de realizar este proyecto es, como se ha esbozado en el apartado anterior, que se pierde mucho tiempo a la hora de desplegar aplicaciones distribuidas, lo cual representa un problema de mayor complejidad e impacto hoy en día debido a la aparición disruptiva del *Cloud Computing*.

En empresas en las que se desarrolla software de forma habitual o se hace uso de software de terceros para la implementación de nuevos productos, como es el caso de Telefónica I+D, es habitual que varios equipos de personas trabajen de forma simultánea en un mismo proyecto desempeñando roles diferentes. A la hora de probar el código de una herramienta, lo más probable es que éste tenga una serie de dependencias que han de ser instaladas previamente, así como unas opciones concretas según el rol de la persona que pretende utilizarlo –no usará para lo mismo el código de una herramienta una persona que se encargue de las bases de datos que una que desarrolle, por ejemplo–. Esto motiva el desarrollo de una herramienta que facilite el trabajo de estas personas y con ello incremente su productividad. Así, cada persona podrá instalar y actualizar estas dependencias con facilidad y tendrá a su disposición las acciones concretas a

realizar según sea necesario, simplificando en gran medida el trabajo humano y agilizando el desarrollo.

Aunque el mercado cuenta con herramientas ya existentes que despliegan aplicaciones de una forma muy completa, las especificaciones que la empresa interesada indicó necesitaba debían ser implementadas de cero cumpliendo unos requisitos concretos, no por ello ignorando los conocimientos que las herramientas previamente citadas podían proporcionar para asentar una base en lo que al despliegue de aplicaciones distribuidas se refiere.

1.2. Objetivos

Como ya se ha mencionado, el objetivo principal de este TFG es el desarrollo e implementación de una solución software con capacidad para el despliegue de aplicaciones distribuidas en un conjunto de máquinas remotas desde una única máquina local.

La lista de objetivos concretos de este trabajo es la que sigue:

- Análisis comparativo de las aproximaciones, funcionalidades y soluciones existentes para el desarrollo y despliegue de aplicaciones distribuidas.
- Estudio de las diferentes posibilidades arquitectónicas y lenguajes de programación a utilizar para la resolución del problema planteado.
- Diseño de la arquitectura del software teniendo en cuenta las diferentes dependencias y la gestión de la configuración distribuida y el entorno necesarios.
- Implementación del sistema completo que permita realizar el despliegue de aplicaciones distribuidas así como acciones referentes a dichas aplicaciones desde una única máquina local utilizando únicamente la herramienta y el código de la aplicación en sí, que deberá cumplir unos requisitos concretos, descritos más adelante..

1.3. Organización del resto del documento

Este apartado se ocupa de esbozar el contenido del resto de capítulos del documento con el propósito de facilitar su lectura y comprensión.

El segundo capítulo trata sobre las tecnologías que se han utilizado para el desarrollo de esta herramienta, desde lenguajes de programación a librerías o recursos web.

El capítulo 3, titulado *Estado del arte* contiene los estudios realizados sobre las tecnologías que existen actualmente y pueden prestar una solución, al menos parcial, al problema que aborda este trabajo, pese a que según requisitos no fueran a usarse. Previamente se hace una introducción sobre aplicaciones distribuidas.

En el capítulo 4 se pasa a analizar más a fondo la prueba de concepto que se realizó a partir de la cual nace este proyecto, el Cli, explicando su estructura, funcionamiento y puntos a mejorar.

A continuación, en el capítulo 5 se describe todo lo referente al desarrollo de la herramienta resultado de este proyecto, Deptool, indicando las diferencias con respecto a la base y la razón de las mismas, los cambios que ha ido sufriendo a lo largo de su desarrollo, el modo de empleo y la funcionalidad, así como los requisitos que se piden para que el usuario que vaya a utilizarla entienda su funcionamiento correcto por completo y consiga sacarle partido. También se encuentran en este capítulo la metodología seguida y las pruebas realizadas.

Se pasa después a un sexto capítulo donde se detallan las conclusiones extraídas tras la finalización del proyecto, así como las líneas futuras a seguir a la finalización de este proyecto.

Seguidamente se encuentran las referencias bibliográficas.

Para finalizar se ofrece un anexo con una tabla que contiene las acciones que pueden realizarse con la herramienta y los diferentes argumentos que acepta cada tarea.

Capítulo 2

Tecnologías utilizadas

A lo largo de este capítulo se explicarán los fundamentos básicos de las diferentes tecnologías y librerías que han sido utilizadas para la creación de la herramienta, empezando por una visión general de los conceptos referentes a IaaS, al tratarse de una herramienta que despliega aplicaciones y las utiliza de forma remota en la nube. A continuación se describen los lenguajes y herramientas detallando finalmente en cada caso las librerías específicas. Se hará así una visión rápida sobre Python, Fabric, Amazon Web Services y Boto.

2.1. IaaS

Como ya se comentó en la introducción, en los últimos años nos hemos encontrado ante el auge del Cloud Computing y su uso en el desarrollo es cada vez más necesario, sobre todo si hablamos de proyectos a gran escala. IaaS es una de las tres categorías principales del servicio de computación en la nube junto con SaaS (*Software as a Service*) y PaaS (*Platform as a Service*).

IaaS (a veces también denominado HaaS, *Hardware as a Service*) es un modelo de disposición en el que una organización proporciona el equipo utilizado para realizar las operaciones, incluido el almacenamiento, el hardware, los servidores y componentes de red. El proveedor de servicios posee el equipo y es el responsable de alojarlo, que éste funcione y su mantenimiento. El cliente paga en función del uso que haga.

Algunas de las características y componentes de IaaS son el modelo de facturación y de *utility computing*, la automatización de tareas administrativas,

el escalado dinámico, la virtualización de los escritorios, los servicios basados en políticas y la conectividad a internet [2].

Actualmente muchas de las grandes compañías que tienen alguna rama en el mundo informático proporcionan estos servicios. En lo que a este proyecto atañe, más adelante en este mismo capítulo profundizaremos en la que se ha utilizado para su desarrollo, Amazon Web Services (AWS).

2.2. Python

Python es un lenguaje de programación de alto nivel interpretado (los programas escritos en este lenguaje son ejecutados por un intérprete) y multipropósito, que puede ejecutarse de dos maneras: en modo interactivo desde la consola (basta teclear *python* para acceder) o en modo script. En la actualidad Python es uno de los lenguajes de programación con mayor proyección, cada vez más usado entre los desarrolladores de software [3].

El origen de Python se remonta a principios de los noventa de mano de Guido van Rossum, un investigador holandés. La primera versión del lenguaje vio la luz en 1991 pero no fue hasta tres años más tarde que decidió publicarse la versión 1.0 [4].

Tras diez años de enseñanza de Python, Mark Lutz cita [5] los siguientes factores como los más populares entre sus alrededor de 200 grupos y 3000 alumnos para demostrar así las principales características del lenguaje:

- Calidad de software: Python se caracteriza por su legibilidad y coherencia. El código en Python está diseñado para ser fácil de leer, y por tanto, de reusar y mantener. La uniformidad del código lo hace fácil de comprender aunque no lo haya escrito uno mismo.
- Productividad del desarrollador: Comparado con el uso de lenguajes compilados o estáticamente tipados como C, C++ o Java, Python aumenta la productividad del programador. El código en Python suele ser entre la tercera y la quinta parte de lo equivalente en C++ o Java, lo que disminuye lo que hay que escribir, depurar y mantener. Se añade a esto además que los programas en Python se ejecutan inmediatamente sin los pasos de compilado. Las *Figuras 3* y *4* ilustran las diferencias en la ejecución entre un lenguaje interpretado y uno tipado [3].

Figura 3. Pasos en la ejecución de un lenguaje interpretado. **Fuente:** *Think Python. How to Think Like a Computer Scientist*, 2012

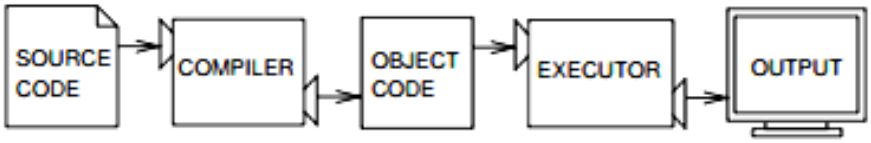

Figura 4. Pasos en la ejecución de un lenguaje compilado. **Fuente:** *Think Python. How to Think Like a Computer Scientist*, 2012

- Portabilidad de los programas: La mayoría de los programas en Python pueden ejecutarse sin cambios en cualquier plataforma o sistema operativo.
- Soporte para librerías: Python viene con una gran colección de funcionalidades predeterminadas en la librería estándar, pero se puede extender con una enorme colección de librerías y aplicaciones externas.
- Integración de componentes: Los scripts en Python pueden comunicarse con otras partes de una aplicación usando una variedad de mecanismos de integración, que permite usar a Python como una personalización de un producto o una herramienta de extensión. Por ejemplo, puede invocar librerías de un lenguaje, integrarse con componentes de otro, comunicarse a través de frameworks e interaccionar a través de redes con diferentes interfaces. No se trata de una herramienta independiente, sino que tiene posibilidades infinitas.
- Disfrute: Aunque este punto se trata de un beneficio intangible, el efecto que este disfrute provoca, aumenta la productividad. La facilidad de uso de Python y sus herramientas integradas convierten el acto de programar en algo más placentero y excitante que rutinario y laborioso.

Por estas y otras razones, Python es un lenguaje altamente utilizado actualmente no sólo por desarrolladores independientes, sino en grandes empresas, como demuestra el caso concreto de este proyecto, ya que Python forma parte del plan tecnológico de Telefónica I+D.

2.3. Fabric

Fabric es una librería para Python y una herramienta de línea de comandos que facilita las tareas de administración de sistemas y permite el despliegue de aplicaciones, la ejecución de comandos y el tráfico de archivos a máquinas remotas. Proporciona además un conjunto básico de operaciones para ejecutar de forma local o remota a través de una consola de comandos (normalmente vía sudo). [6][7]. La revista Linux Journal publicó el pasado año un artículo en el que afirmaban que Fabric era "el mejor amigo de un administrador de sistemas" [8].

En el marco de realización de este proyecto, Fabric será el encargado de realizar toda comunicación con las máquinas remotas, tanto las que se crean como las que se indiquen en la configuración previamente. Se podrá gracias a esta librería no sólo crear nuevas máquinas o cambiar el estado de las mismas, sino también instalar dependencias o software como si de nuestra propia máquina local se tratara.

Para la utilización de Fabric se necesita crear un archivo llamado fabfile que contendrá una serie de funciones escritas en Python denominadas tareas (*tasks*), que serán aquellas que se podrán invocar y ejecutar desde la consola de comandos.

Una vez completo el fabfile podremos llamar a cada una de las funciones que en él se encuentran implementadas usando la línea de comandos *fab.* Un ejemplo muy sencillo de fabfile [7], sería, por ejemplo, el que se muestra en la *Figura 5.*

```
from fabric.api import run

def host_type():
    run('uname -s')
```

Figura 5. Ejemplo de fabfile. **Fuente:** *Fabric*, 2014

El método más común para la utilización de Fabric es vía *fab*, como se acaba de comentar. El uso básico de *fab* consiste en ejecutar las tareas que se encuentran detrás del comando en orden de lectura, pero este uso puede expandirse añadiendo opciones y/o pasando argumentos a tareas individuales [9]. Puede observarse un ejemplo de ejecución de un fabfile con argumentos en la *Figura 6* [7].

```
$ fab -H localhost,linuxbox host_type
[localhost] run: uname -s
[localhost] out: Darwin
[linuxbox] run: uname -s
[linuxbox] out: Linux

Done.
Disconnecting from localhost... done.
Disconnecting from linuxbox... done.
```

Figura 6. Ejecución con argumentos del fabfile de la Figura 5. **Fuente:** *Fabric*, 2014

Además de ejecutar las diferentes tareas del fabfile, podremos importarlas junto con otros elementos de Fabric para que pueda ser utilizado como funciones normales de Python.

Una de las principales ventajas que proporciona Fabric es la posibilidad de ejecutar las tareas definidas en el fabfile en una lista de máquinas, que pueden ser especificadas de diversas maneras, reduciendo enormemente el tiempo que se tardaría en realizar las acciones máquina por máquina. Además, el sistema de roles con el que cuenta facilita aún más este cometido, pudiendo especificarse tareas para un rol específico, consiguiendo así la ejecución de tareas concretas en toda máquina asociada a ese rol.

Para añadir una capa de seguridad al conectarse a máquinas remotas que estarán asociadas a un usuario que las financia, Fabric utilizará una ruta que habrá de proporcionársele para conectarse vía ssh con la máquina, asegurando así que sólo quien tenga acceso a dicho certificado puedan acceder a las máquinas.

2.4. Amazon Web Services

Amazon cuenta con una larga trayectoria usando una infraestructura descentralizada, observando entre sus filas de desarrolladores un aumento de la productividad y agilidad. Para 2005 Amazon llevaba más de una década construyendo y gestionando una infraestructura a gran escala fiable y eficiente para dar servicio a uno de los mayores servicios de comercio online, por lo que, para que otras organizaciones pudieran beneficiarse de la experiencia la inversión (millones de dólares), Amazon creó Amazon Web Services (AWS) en

2006, sirviendo esta plataforma en la actualidad a cientos de miles de clientes [10].

AWS es una plataforma completa de servicios en la nube que ofrece potencia de computación, almacenamiento, entrega de contenidos y mucha más funcionalidad que puede ser utilizado por las organizaciones clientes para desplegar aplicaciones de manera rentable, flexible, fiable y escalable. La estructura por servicios de AWS es la que se muestra en la *Figura 7* [10], de los cuales para este proyecto se han utilizado y pasaremos a detallar EC2, S3 y SES.

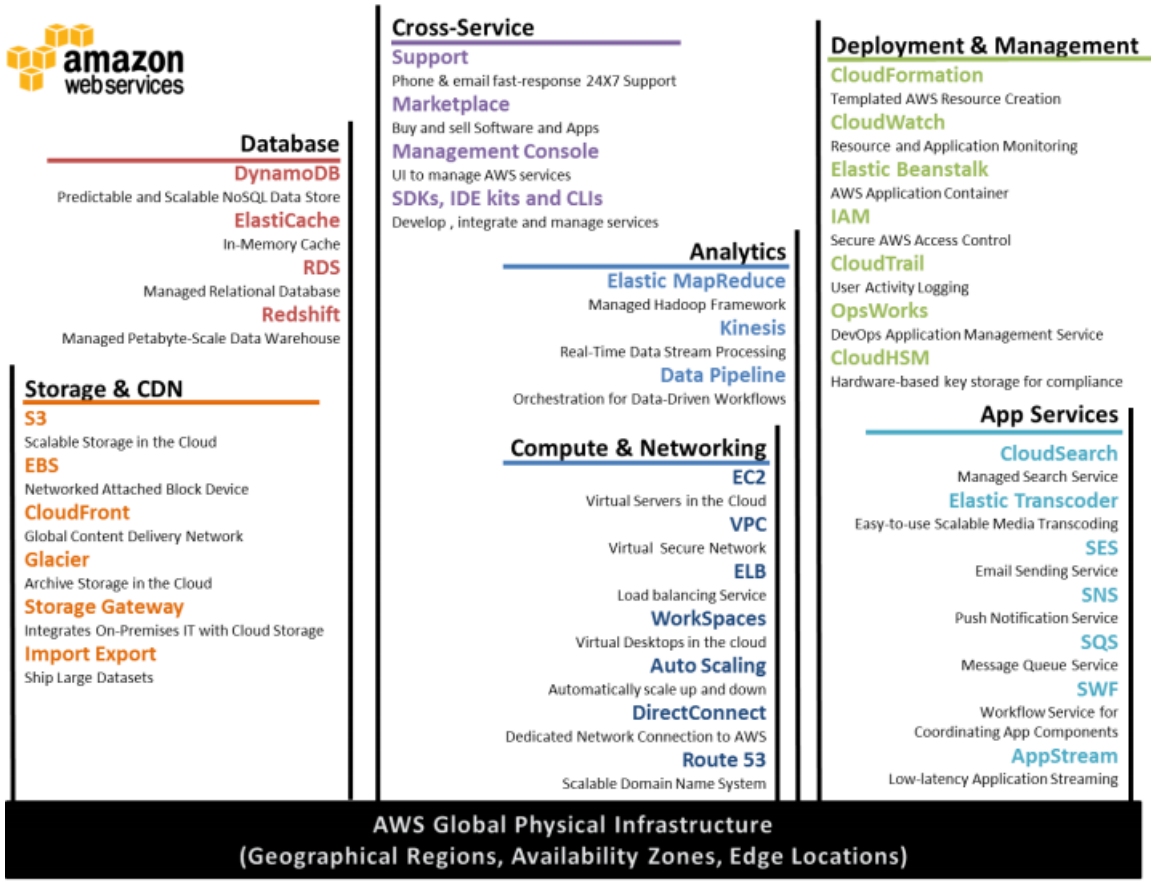

Figura 7. Estructura de servicios de AWS. **Fuente:** Amazon Web Services, 2014

2.4.1. EC2

Amazon Elastic Compute Cloud (Amazon EC2) es un servicio web que proporciona capacidad informática con tamaño modificable en la nube.

Cuenta con una interfaz muy sencilla que proporciona de forma muy visual control completo sobre los recursos proporcionados. Cuenta además buenas medidas de seguridad y privacidad.

Para usar EC2 el cliente crea una AMI (una imagen de máquina de Amazon) que contiene el sistema operativo, los programas de aplicación y las opciones de configuración. A continuación, esta AMI se carga en S3 (se explicará más adelante) y se registra en EC2 creando un identificador de AMI (AMI ID). Teniendo esto, el cliente puede crear máquinas virtuales a partir de esa AMI

En este proyecto se hace uso de EC2 para la creación de máquinas remotas y las operaciones que posteriormente se realizan sobre ellas.

2.4.2. S3

Amazon Simple Storage Service (Amazon S3) es, como su propio nombre indica, almacenamiento para internet.

Su sencilla interfaz de servicios web puede usarse para almacenar y recuperar la cantidad de datos que se desee en cualquier momento y desde cualquier parte de la web.

En este proyecto se hace uso de S3 cuando se crean nuevas AMIs desde la herramienta, ya que éstas, como se ha explicado en el apartado anterior, deben almacenarse en S3 cuando se crean.

2.4.3. SES

Amazon Simple Email Service (Amazon SES) es un servicio de envío de correo electrónico exclusivamente de salida.

En este proyecto se ha utilizado para enviar correos a los usuarios de una lista predefinida y validada en AWS al término de la creación de una AMI, ya que el tiempo que ha de transcurrir en esta operación es mucho mayor que el de las demás.

2.5. Boto

Boto es una librería en Python para Amazon Web Services con la que se realizan fácilmente las llamadas pertinentes a las APIs correspondientes para, por ejemplo, establecer conexión o crear los diferentes recursos que Amazon Web Services proporciona.[11]

Se ha utilizado en este proyecto por la facilidad que Boto presenta a la hora de programar estas llamadas en Python.

Capítulo 3

Estado del arte

Primeramente, realizaremos en este apartado una pequeña introducción general de qué es y en qué consiste una aplicación distribuida para más adelante profundizar en cada una de las herramientas existentes que ayudan al despliegue de este tipo de aplicaciones y han servido de base de conocimientos para el desarrollo de nuestra herramienta.

3.1. Aplicaciones distribuidas

A principios de la década de los 80 se diseñó en la Universidad de California en Berkeley una interfaz para BSD Unix que permitía a los programas acceder y comunicarse a través de una red de comunicaciones. Desde entonces, la necesidad de los desarrolladores de utilizar aplicaciones distribuidas como si fueran locales no ha hecho más que crecer.

Una aplicación distribuida es aquella en la que los distintos componentes se ejecutan en entornos separados, generalmente diferentes plataformas conectadas a través de una red.

Entendemos como componente cada unidad independiente que forma un sistema más completo, cada uno de los cuales tiene asignado un conjunto específico de responsabilidades dentro de un sistema. La distribución, como ya se ha dicho, indica que los diferentes componentes se encontrarán en máquinas diferentes, estableciendo así una separación física de dichos componentes (separación en niveles —*layers*—). Cuando se hable de la distribución lógica en lugar de la física se usará el término capas —*tiers*—.

Con esta distribución se busca cubrir una serie de necesidades, tales como ofrecer la posibilidad de trabajar a distancia y en diferentes lugares o la compartición de información, seguridad en la protección de la información.

Los tipos de aplicaciones más comunes son las de dos niveles (cliente-servidor), las de tres niveles (cliente-middleware-servidor) y las multinivel. Puede observarse un ejemplo de la estructura de una aplicación distribuida multinivel en la *Figura 8* [12].

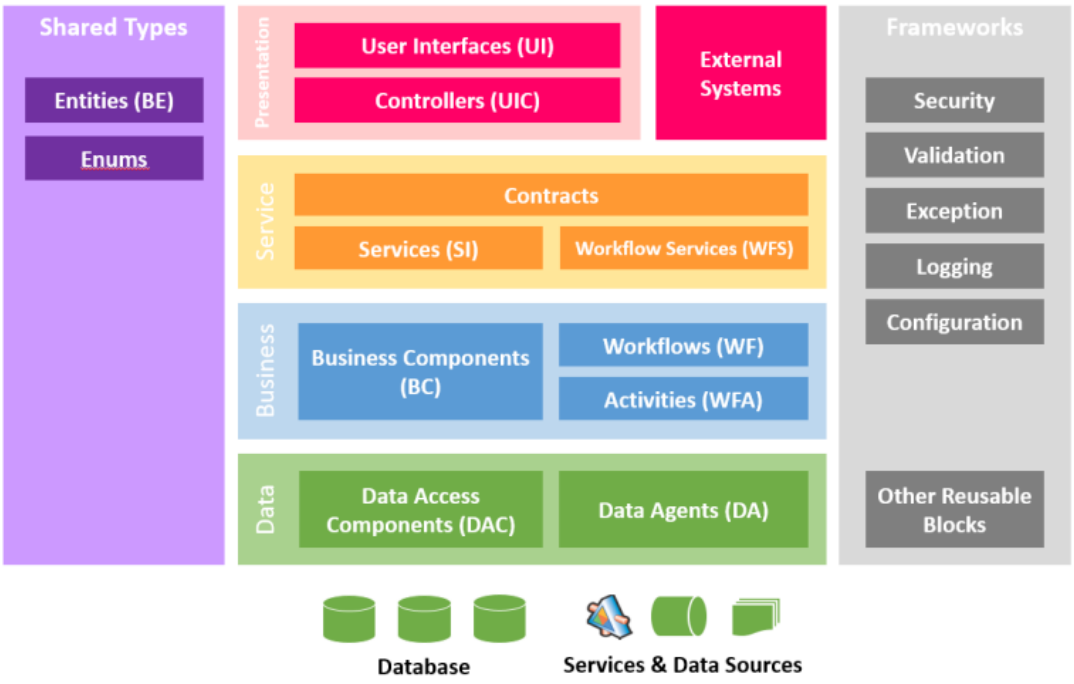

Figura 8. Ejemplo de estructura multinivel para una aplicación distribuida. **Fuente:** Microsoft Malaysia. *MSDN Blogs*, 2013

El diseño de aplicaciones separadas por niveles y/o capas promueve el concepto de separación y ayuda a clasificar juntos en un mismo nivel partes cuyo código o responsabilidad dentro de la aplicación es similar, facilitando así la distribución de los diferentes componentes, pasando así de una separación meramente lógica una física sin mucha dificultad.

El mayor reto de las aplicaciones distribuidas es conseguir que no se note que lo son, esto es, que el usuario no note diferencia entre una versión local de la misma y una que, por ejemplo, almacene sus datos en una máquina remota tras establecer conexión, aunque no conozca su ubicación física [13][14].

3.2. Herramientas

3.2.1 Chef

Chef es una plataforma de automatización de sistemas e infraestructuras cloud que facilita el despliegue de servidores y aplicaciones a cualquier localización física, virtual o en la nube, sin importar el tamaño de la infraestructura. Se usa para acelerar el despliegue de aplicaciones.

Cada organización está compuesta por una o más estaciones de trabajo (*workstations*), un único servidor, y cada nodo (máquinas a configurar con Chef) es configurado y mantenido por el cliente Chef. Los Cookbooks y recetas se usan para indicar al cliente chef cómo se debe configurar cada nodo en la organización. El cliente Chef (instalado en cada nodo) hace la configuración real.

Chef se basa en una idea fundamental: el usuario puede modelar la infraestructura y las aplicaciones como código, sin que Chef haga ninguna suposición sobre en entorno y enfoque que se usará para configurar y manejar éstas. En lugar de esto se proporciona una forma de automatizar la infraestructura y los procesos, convirtiéndolos en 'testeables', repetibles y versionados.

Las definiciones y recetas reusables contenidas en los Cookbooks están escritas en el lenguaje de programación Ruby. El despliegue de las infraestructuras descrito en estos Cookbooks se encuentra detallado en las diferentes acciones a realizar indicando cómo debe desplegarse, configurarse y manejarse cada parte. Aplicando estas definiciones en el servidor, Chef produce y automatiza la infraestructura. Estos Cookbooks y recetas están compuestos de bloques denominados recursos. Muchos de estos recursos están ya incluidos en Chef y pueden encontrarse en la comunidad Chef, pero un usuario también puede crear los suyos propios (teniendo asimismo la opción de subirlos a la comunidad para que sean reutilizados).

El servidor de Chef almacena los datos de la configuración de red (tanto la actual como la deseada) y las recetas, y gestiona los nodos que forman la infraestructura. Los datos describen los "ingredientes" que componen la infraestructura. Las recetas son las instrucciones paso a paso para juntar estos ingredientes y formar un sistema en funcionamiento completo.

Por otra parte, el cliente Chef es un programa que ejecuta las recetas en nodos de la red (tanto físicos como virtuales o en la nube). Además, se usa una estación de trabajo (*workstation*) para actualizar el estado del servidor Chef cada cierto tiempo según la infraestructura evoluciona, siendo todos estos cambios capturados usando un control de revisiones. En esta estación de trabajo además es donde se crean y modifican los Cookbooks, que están almacenados en un repositorio de código base [15][16][17].

Todos estos componentes siguen el esquema presentado en la *Figura 9* [17].

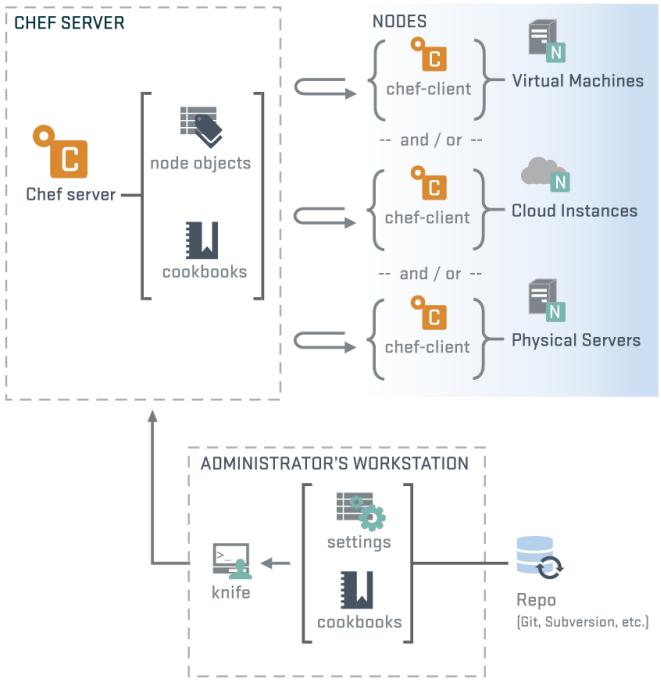

Figura 9. Esquema de componentes de Chef. **Fuente:** *Chef*, 2014

3.2.2 Puppet

Puppet es un enfoque declarativo basado en modelos para la automatización de las TI, que ayuda a gestionar infraestructura durante su ciclo de vida, desde el aprovisionamiento y la configuración hasta la orquestación y los reportes. Con el uso de Puppet se pueden automatizar fácilmente tareas repetitivas, desplegar rápidamente aplicaciones críticas y gestionar el cambio proactivamente, escalando los servidores de decenas a miles en las instalaciones o en la nube.

Funciona siguiendo una serie de pasos (ver *Figura 10* [18]) consistentes en:

1. Definir la configuración de la infraestructura deseada con el lenguaje declarativo de Puppet
2. Simular los cambios en la configuración antes de forzarlos
3. Realizar estos cambios desplegando el estado deseado automáticamente corrigiendo cualquier posible desviación en la configuración
4. Informar sobre las diferencias entre el estado real y el deseado, así como los cambios realizados para llegar al estado deseado

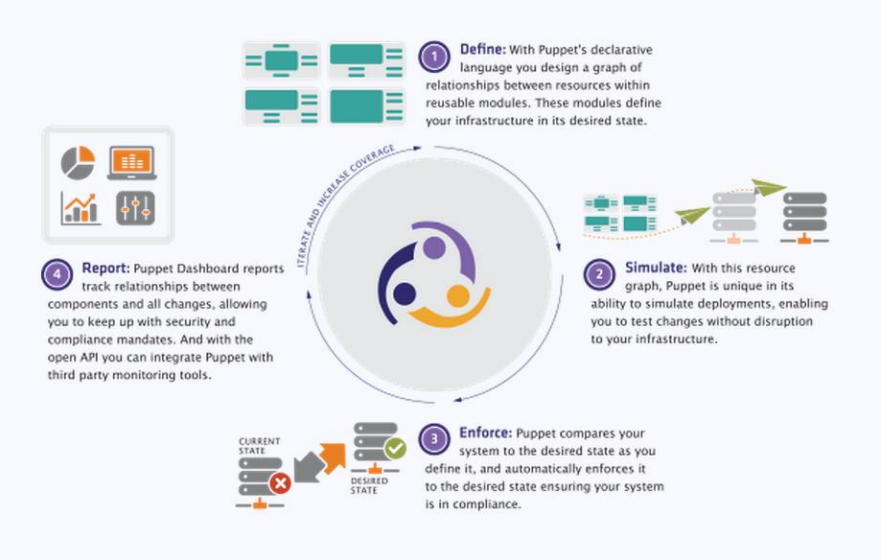

Figura 10. Funcionamiento de Puppet. **Fuente:** *Puppet*, 2014

La definición del estado deseado puede obtenerse descargando uno de los módulos de configuración pre-construidos que se encuentran en el mercado online de Puppet Labs o en Puppet Forge (comunidad de usuarios), o construir una personalizada utilizando el lenguaje de configuración de Puppet, pudiendo más adelante reutilizarse en cualquier entorno o sistema operativo.

Tras desplegar los módulos de configuración, el Agente Puppet se comunica con el Maestro Puppet para forzar automáticamente el estado deseado a los nodos. Para ello el Agente Puppet de un nodo envía al Maestro Puppet los datos sobre su estado (*Facts*), que el Maestro utilizará para recoger los datos sobre cómo debería ser configurado el nodo (*Catalog*) y enviárselos de nuevo al Agente, que realizará los cambios pertinentes para llegar a ese estado (o los simulará si se da el caso) y le enviará al Maestro un reporte completo (*Report*) que será accesible a través de APIs públicas para su integración con otros sistemas [18][19] (ver *Figura 11* [18]).

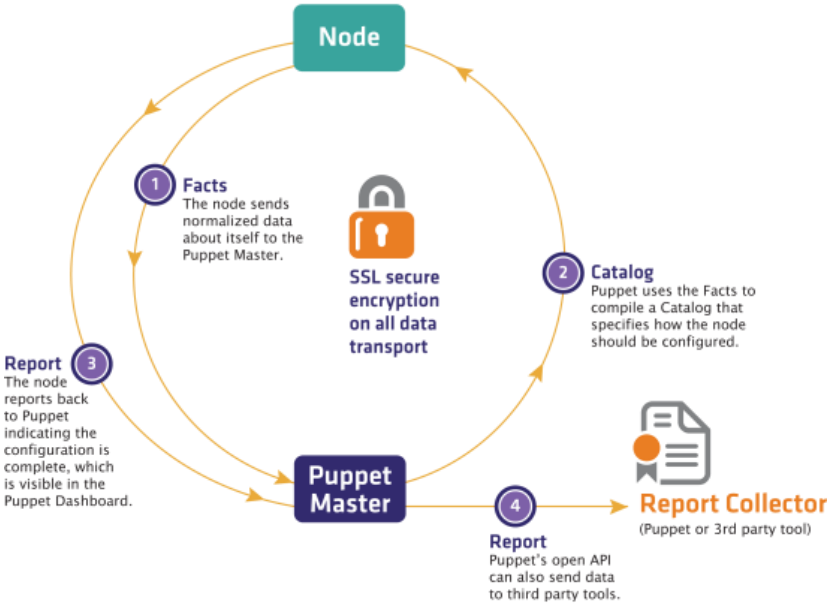

Figura 11. Obtención del estado deseado en un nodo de Puppet. **Fuente:** *Puppet*, 2014

Dentro de las herramientas proporcionadas por el grupo Puppet Labs, la más adecuada a solventar los problemas que a este proyecto atañen, y por tanto, la que se ha investigado más a fondo es Puppet Enterprise. Puppet Enterprise es una plataforma completa de gestión de configuración con un conjunto de componentes optimizado y probado para trabajar bien juntos (para más información, ver [20]). Combina Puppet, una consola web para analizar los reportes y controlar la infraestructura, potentes características de orquestación, herramientas de aprovisionamiento para nube y soporte profesional. Con Puppet Enterprise los administradores de sistemas pueden automatizar tareas repetitivas con facilidad, desplegar rápidamente aplicaciones críticas y controlar proactivamente infraestructura (tanto en algún tipo de instalaciones como en la nube).

3.2.3 Capistrano

Capistrano es una herramienta de automatización de servidor remoto en lenguaje Ruby que se utiliza, entre otras cosas, para desplegar aplicaciones web simultáneamente en cualquier número de máquinas, para manejar herramientas que aprovisionamiento de infraestructuras como Chef (descrito anteriormente en este mismo capítulo) o automatizar tareas comunes en equipos de software.

Una característica importante de Capistrano es su facilidad para ser usado por medio de scripts, pudiendo éstos ser integrados en cualquier otro software que use Ruby pudiendo formar una herramienta mayor o más completa.

Funcionalidades añadidas a Capistrano son, por ejemplo, formateadores de salida intercambiables, facilidad de adición de soporte para software externo, compatibilidad con Rails o soporte para entornos complejos [21].

A grandes rasgos, Capistrano ofrece funcionalidad muy similar en Ruby a la que podemos encontrar con Fabric para Python pero ofreciendo una mayor compatibilidad con otras herramientas, al menos a nivel general.

Capítulo 4

Estudio del Cli

Cli es la prueba de concepto de una herramienta de despliegue de aplicaciones distribuidas a partir de la cual se realiza la que en este proyecto se presenta. Cli fue presentada como una primera versión de funcionalidad muy básica y limitada, ya que con ella se pueden iniciar máquinas pero sólo se realizan acciones sobre ellas en una (la última creada o la especificada a mano en el fichero de configuración).

A continuación se explica el funcionamiento de esta herramienta así como la estructura que presenta.

4.1. Estructura

La principal característica de la estructura de Cli es que se encuentra separado en varios roles ampliables, siendo estos por el momento: admin (administrador), code (código) y db (base de datos).

Con ficheros particulares para cada rol, Cli se divide en las siguientes secciones:

Config

Encontraremos aquí un fichero de configuración por cada rol que contiene los datos que se necesitan para el despliegue de una máquina virtual en AWS, tales como la AMI ID, el tipo de instancia que se desea crear y a qué grupo de seguridad deberá pertenecer, así como el usuario con el que vamos a crearla. En el caso del rol db encontramos también campos para el nombre, usuario y contraseña de la base de datos, asumiendo que toda máquina que se cree desde

este rol será una base de datos, no pudiendo así asignar a otra máquina creada con anterioridad esta funcionalidad.

Todos los roles cuentan además en su archivo de configuración con campos vacíos a rellenar al crearse la máquina virtual con la url de la misma y la ID única de la instancia creada.

Además de estos ficheros encontramos uno de configuración general del Cli en el que encontraremos las claves de acceso y características generales de Amazon, así como las direcciones web del repositorio de la aplicación con la que trabajaremos (incluyendo la rama con la que queremos trabajar).

Dependencies

Dentro de esta sección se encuentran una serie de ficheros ejecutables en Bash que instalarán para el rol seleccionado una serie de dependencias para un sistema operativo determinado (acordado previamente para la realización de los archivos y la creación de las máquinas) que posteriormente necesitaremos tener instaladas en la máquina virtual al realizar las acciones pertinentes. En esta versión sólo se contaba con un fichero ejecutable para CentosOS.

Util

En la sección Util se encuentran los ficheros con funciones para el control de la configuración. Para esta prueba de concepto se utilizaron únicamente funciones básicas que controlaban la lectura y escritura de los diferentes campos de los ficheros de configuración, así como una opción para incluir estos campos en el PATH de la máquina local.

Fabfiles

Como su propio nombre indica, aquí se encuentran los diferentes ficheros de Fabric que contienen las acciones a realizar por cada rol.

Desde el archivo Fabric del admin pueden instalarse todas las dependencias indicando el rol del cual se deseen instalar. Este mismo rol es el único que podrá

arrancar, parar o eliminar máquinas virtuales. Las características únicas del rol de código consisten en la clonación del repositorio donde se encuentra el código de la aplicación a desplegar y con la que trabajar, la actualización de dicho código, una función que ejecuta este código y una que muestra los logs generados, además de una parada forzosa del mismo. El rol correspondiente a la base de datos tiene como funciones características aquellas que la inician, crean una, un usuario o dan permisos.

Función principal

Aparte de esta estructura encontramos un fichero que sirve de inicio, en el que se indican los argumentos correctos para la llamada a la herramienta desde consola y se forma la llamada particular de la herramienta para simplificar los mandatos predeterminados de Fabric.

4.2. Funcionamiento

Sirviendo como prueba de concepto en mayor parte para probar las capacidades de Fabric, Cli se mostró útil para probar que se podía crear de forma sencilla una herramienta capaz de desplegar e instalar máquinas remotas desde una máquina local sin necesidad de conectarse manualmente a esas máquinas remotas.

Al término de esta primera versión de la herramienta, Cli era capaz de iniciar instancias en Amazon Web Services y realizar acciones sencillas en ellas de una en una, o bien en la última creada o bien en la que esté especificada en los ficheros de configuración.

La llamada correcta a la herramienta debía realizarse utilizando al menos dos argumentos, el primero de los cuales se correspondía al rol desde el que se realizaba la acción, y dicha acción, que se correspondía con el segundo argumento. Dependiendo de cuál fuera esta acción, los argumentos de después variaban.

4.3. Mejoras

Aunque cuando se presentó, la funcionalidad conseguida fue más que suficiente para lo que se esperaba, la necesidad de una mejor gestión de las capacidades de AWS y Fabric en el despliegue y automatización de tareas en una lista de máquinas virtuales en lugar de una sola dio lugar a este proyecto.

Además, otra gran necesidad debía ser cubierta en lo referente a la gestión y coordinación de la configuración de las máquinas y aplicaciones, pero separando todo que tenga que ver con estas acciones de la herramienta original, ya que dejarían de ser responsabilidad del desarrollador o administrador encargado del despliegue, pasando a ser recibida desde un lugar centralizado. Al tratarse de una idea independiente, dio lugar a otro proyecto paralelo a este y desarrollado por otra persona del mismo laboratorio.

Capítulo 5

Desarrollo

A lo largo de este capítulo se detallan todos los pasos que se han seguido en el desarrollo de la herramienta resultante de este proyecto, se detalla su estructura, funcionamiento y diferentes opciones de ejecución y se comenta la metodología seguida así como las pruebas realizadas.

5.1. Introducción

Una vez presentado el Cli (descrito en el capítulo 4) como resultado de una prueba de concepto, surgió este proyecto como idea para mejorar ese primer esbozo de herramienta con el que se desplegaban aplicaciones en una máquina virtual previamente creada todo desde una máquina local. Dicha prueba de concepto, con la que se demostró que la funcionalidad deseada podía ser implementada de forma mucho más sencilla que utilizando una herramienta externa como las descritas en el capítulo 3, pese a cumplir las expectativas de un primer intento, carece de características esenciales en una herramienta de este tipo, por lo que, mejorando las fuentes con las que se contaban, en este proyecto nace Deptool, una segunda versión de Cli.

5.2. Cambio de planteamiento

Una vez estudiado a fondo el funcionamiento y la estructura del Cli y habiendo explorado más a fondo la funcionalidad que Fabric puede aportar, se decidió cambiar el planteamiento que en principio presentaba Cli, el modelo separado por roles.

Los motivos que llevaron a tomar esta decisión fueron que, por una parte, Cli mezclaba en un mismo concepto de forma no lo suficientemente clara rol de persona, como puede ser el de admin, y rol de máquina, como era el de db, cuando además, Fabric ya cuenta con un concepto propio de rol para separar las acciones de uno u otro (refiriéndose a máquinas) sin necesidad de separar ficheros.

Además, aunque se pierde la comodidad de poder añadir un nuevo rol con toda su funcionalidad, dependencias y configuración de forma sencilla con archivos diferentes, se gana en sencillez con la utilización de un único fabfile, los ficheros de dependencias que se deseen y una configuración general acompañada del mismo modo por un único fichero adicional por cada aplicación, aunque esto se explicará más a fondo más adelante en este mismo capítulo.

En el estudio del Cli se observó además que algunas de las tareas a realizar eran muy específicas de un proyecto en concreto, no generales para una herramienta de este estilo como se supone deberían ser. Se centraban en un proyecto en Python en concreto, no dejando opción a que la herramienta funcionara con otros. Se ha decidido suprimir así algunas de las tareas, sobre todo del rol code, las cuales sin conocer previamente con qué aplicación se utilizará la herramienta son imposibles de implementar de forma general.

El resto de cambios en esta herramienta con respecto a la prueba de concepto inicial se centran en la incorporación de nuevas funcionalidades y en la extracción de funciones auxiliares comunes a un fichero independiente, pero esto se detallará más adelante.

5.3. Diseño

La fase de diseño se centró en la primera parte del proyecto, cuando se decidieron los cambios y mejoras a realizar a partir del Cli aprovechando mejor de lo que éste lo hacía, las funcionalidades que Fabric proporciona.

En primer lugar, debía conseguirse cambiar el enfoque basado en roles (ver *Figura 12a*) a uno más generalista y rápido con un único fabfile (ver *Figura 12b*) y pasar de operar en una máquina (ver *Figura 13a*) a hacerlo en todas las indicadas en la configuración (ver *Figura 13b*) y

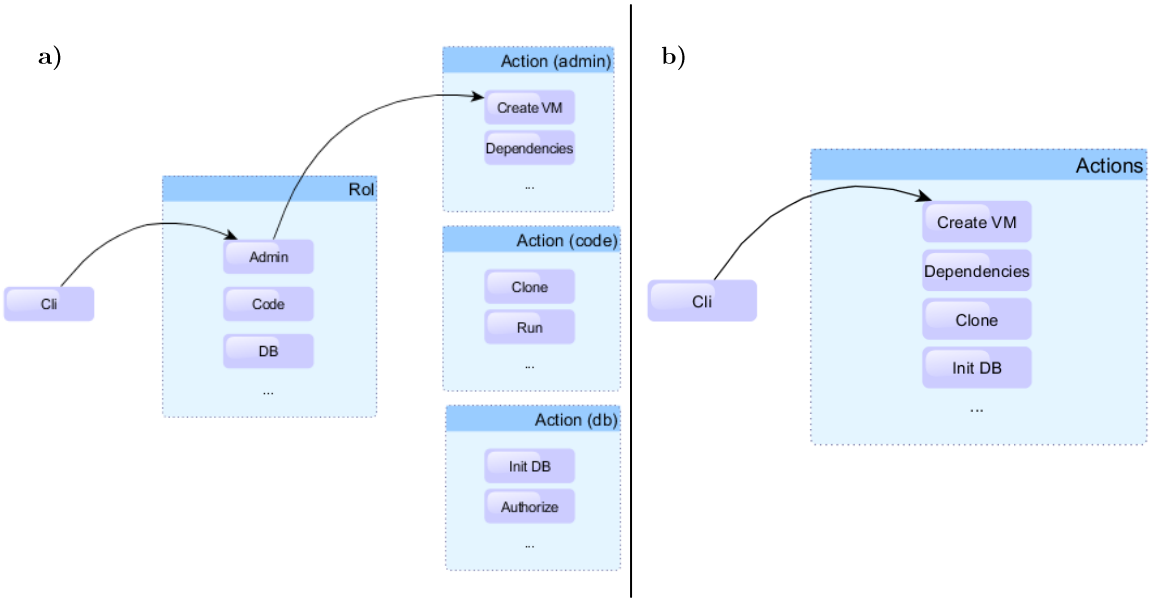

Figura 12. Modelo inicial de ejecución **(a)** frente al modelo ideal de ejecución para Deptool **(b)**

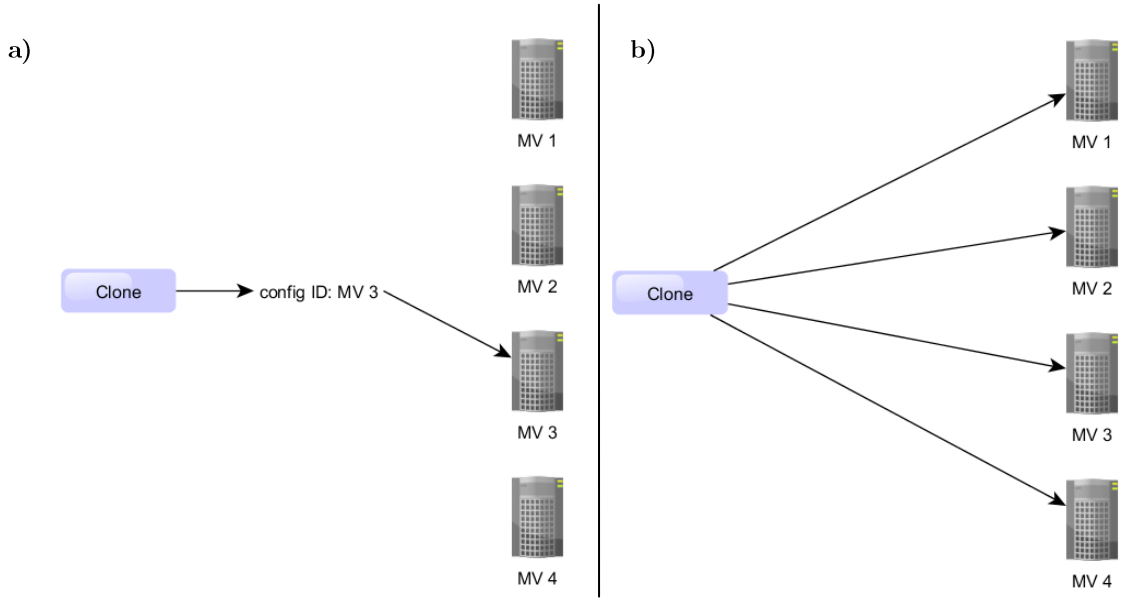

Figura 13. Funcionamiento inicial del Cli sobre las máquinas **(a)** frente al funcionamiento ideal de Deptool sobre las máquinas **(b)**

En segundo lugar y con este diseño claro en mente, se estudiaron las diferentes opciones para gestionar de forma sencilla la configuración, de forma que se gestionara de la forma más sencilla posible de una forma correcta con esta

herramienta pero no fuera difícil adaptarla en un futuro usando otra que realizara una gestión y control más completos. Se decide así lo siguiente:

- El formato de los archivos de configuración cambia de .env con formato de declaración de variables de entorno a .json, más fácil y rápido de interpretar. Con esto además se consiguen suprimir muchas funciones de parseo de variables y de copia en el PATH de la máquina al poderse leer con funciones Python y utilizar en entorno que proporciona el mismo Fabric. Se decidió en este punto también cómo y dónde se formarían los host completos, necesarios para la ejecución de la herramienta pero imposibles de proporcionar antes de crearse las máquinas remotas.
- La herramienta tendrá un archivo de configuración general y uno para cada aplicación que quiera usarse desde la ella. Tenemos así dos únicos archivos de configuración para cada ejecución con todo lo necesario, evitando posibles conflictos con los que pudiera haber en los diferentes roles.
- Al final de toda operación que toque algún valor de la configuración, ésta se actualizará en su archivo correspondiente, no sólo en el entorno en el momento de la ejecución. Aunque esta acción parece trivial, se observó que en el Cli no se realizaba, perdiendo así parte de los cambios. En un futuro esta acción deberá ser revisada utilizando la herramienta de control de configuración realizada en otro proyecto, ya que los archivos de configuración permanecen siempre en la máquina local y es posible que quieran ser utilizados por otra persona del equipo en sus despliegues.

Del mismo modo, se cambió el modelo de presentación de dependencias. De tener un fichero por rol se pasa a los que el desarrollador considere pertinente en función de cómo tenga separadas estas dependencias, siendo estos ficheros proporcionados por él, teniendo en cuenta que sabrá el sistema operativo concreto en el que se instalarán, que vendrá indicado por la AMI desde la que se han creado las máquinas.

Se decidió también en la fase de diseño qué operaciones debería ser posible realizar desde la herramienta dado el uso más general que queríamos darle, suprimiendo acciones demasiado concretas presentadas en el Cli, que aunque para un proyecto en concreto daban una buena solución, no servían para una aplicación cualquiera, como se había planteado en la propuesta del proyecto. Las

operaciones que se eligieron para formar parte de la herramienta se detallan más adelante en este mismo capítulo (*5.6. Opciones*).

5.4. Estructura

Una vez modificado el enfoque estructural de Cli para el desarrollo de Deptool, se propone una nueva estructura no muy alejada de la anterior pero igual de modular y sencilla de controlar. Se mantienen así algunas de las secciones que encontrábamos en el Cli pero desaparecen otras que no aplican.

La estructura final de la herramienta pasa a ser la siguiente:

Fabfile y archivo principal

Con el nuevo concepto sin roles, el uso de una sección exclusiva para el almacenamiento de los diferentes fabfiles hacía la herramienta más compleja de lo que debería, por lo que se decidió tener tanto el fabfile (que contiene todas tareas disponibles) como el archivo principal de la herramienta en la raíz.

El archivo principal no consistirá más que en un validador de tareas que imprime un error en caso de introducir una incorrecta o que no puede ser utilizada dado el estado actual, y hace que la configuración sea cargada antes de la ejecución de cualquier tarea para mantenerla actualizada. Se elimina la función que creaba la llamada principal para usar mandatos de Fabric que faciliten este cometido y no cambiar así su política. El funcionamiento de la herramienta se detalla más adelante.

El fabfile en este caso contiene todas las tareas que pueden realizarse con la herramienta. Estas tareas se especificarán en el apartado *Opciones* de este mismo capítulo.

Config

Como se ha indicado en la fase de diseño, la nueva sección de configuración se compone de, al menos, dos ficheros, uno de ellos con la configuración general de la herramienta con los datos del usuario, y otro con la configuración específica de la aplicación con la que se va a trabajar.

En el fichero *config.json,* correspondiente a la configuración general, podremos encontrar la información referente a la cuenta de AWS que utilizaremos, así como las opciones que deseamos utilizar para la creación de nuevas máquinas y los datos de la aplicación que utilizaremos posteriormente (nombre, repositorio y rama). Por último encontramos aquí la lista de correos a quien deberán enviarse en caso de ser necesario, con el asunto del mismo y el remitente.

El fichero de conjuración de la aplicación, *app.json*, por su parte, contiene los datos específicos para las máquinas creadas para esa aplicación, para la creación de bases de datos desde esas máquinas y los diferentes hosts que se forman al crear las máquinas que se correspondan con la aplicación.

El contenido concreto de cada campo de ambos ficheros se explicará con detalle en el apartado de *Requisitos* de este mismo capítulo.

Dependencies

La primera decisión que se tomó con respecto a las dependencias fue que cada aplicación tuviera su propio archivo de dependencias, tal y como se hace con la configuración, pero pensando en que los desarrolladores puedan separarlas según su propio criterio, se optó por la opción de que existieran tantos ejecutables como se deseara, ya que se instalará cuando llegue el momento únicamente el que se especifique. Esto no impide que se tengan todas las dependencias en un único fichero si se desea, sino que da libertad para tenerlas según gustos, pudiendo separarlas según las operaciones que se vayan a realizar a continuación, evitando así la instalación innecesaria de dependencias en las máquinas.

La persona que proporcione estos archivos deberá tener en cuenta que según sea una u otra la AMI seleccionada para crear las máquinas, el ejecutable de dependencias deberá ser el adecuado para ese sistema operativo o se recibirá un error. La operación de instalación de dependencias se detallará más adelante en este mismo capítulo.

Utils

Al cambiarse el planteamiento en cuanto a configuración, el contenido de la sección debería haberse eliminado, pero finalmente se aprovechó para incluir aquí todas las funciones auxiliares que se utilizaban en las deferentes tereas.

La mayoría de estas funciones consisten en las operaciones sencillas que se realizan por medio de Boto para interaccionar con AWS, tanto con EC2 como con S3 y SES. Como extra se añadió en este fichero de funciones auxiliares una que validara los nombres para la creación de nuevas AMIs y una que cargara y preparara la configuración para empezar a usar la herramienta.

5.5. Requisitos

Para la utilización de la herramienta es necesario que la persona que vaya a darle uso haga cumplir una serie de requisitos previos para no obtener ningún error. Por un lado, debe proporcionar los ficheros de configuración y dependencias y colocarlos en sus respectivas carpetas; por otro, los ficheros de configuración deben seguir el siguiente patrón:

- El fichero de configuración de la aplicación que se vaya a utilizar debe llamarse *app.json* y contar con los siguientes campos:

```
{ "aws":{
    "ami_id":" " ,
    "instance_type":" ",
    "security_group":" "
  },
  "db":
    { "name":" ",
      "pass":" ",
      "user":" "
    },
  "hosts":[
    {
      "aws_instance_id":" ",
      "url":" ",
      "user":" "
    },
    { … }
  ],
  "user":"ubuntu"
}
```

Figura 14. Estructura correcta del fichero *app.json*

Dentro de los referentes a AWS, deberán estar especificados el identificador de la AMI a partir del cual se van a crear las máquinas para la aplicación (éste será sustituido automáticamente si se crea una nueva desde la herramienta), el tipo de instancia de EC2 que se deberá crear y el grupo de seguridad al que deberán pertenecer.

Con respecto a las posibles bases de datos que se creen, deberán estar especificados en la configuración un nombre para la misma y sus usuario y contraseña. Estos sólo se utilizarán en caso de realizar la operación completa de inicio de una base de datos, en casos particulares podrán ser especificados por el usuario.

Deberá especificarse un usuario por defecto para la creación de las máquinas (p. ej. ubuntu). Si se cuenta con alguna máquina ya creada para la ejecución de la herramienta, el campo *hosts* deberá contar con, al menos, uno, que contendrá el identificador de la máquina, la url y el usuario. En caso negativo, este campo estará vacío y se crearán los diferentes host de forma dinámica según se creen máquinas. En caso de encontrarse vacío y seleccionar otra tarea que no sea la creación de una máquina se recibirá un error.

- Debe rellenarse el fichero de configuración general de la herramienta (*config.json*) con los datos oportunos, siendo estos:

```
{  "aws":{
     "access_key_id": " ",
     "secret_access_key": " ",
     "key_pair": " ",
     "region": " ",
     "ami_id": " "
   },
  "bucket_name": "",
  "pem": " ",
  "repo":
    {  "name": " ",
       "url": " ",
       "branch": " "
    },
  "mail":
    {  "recipients": " ",
       "subject": " ",
       "source_email": " "
    }
}
```

Figura 15. Fichero *config.json* vacío

En lo referente a AWS, deberán rellenarse las claves del cliente de AWS que se va a utilizar, así como el nombre de la *key pair* y el identificador de la AMI a partir de la cual se crearán las máquinas por defecto. El nombre del *bucket* donde se alojarán las AMIs creadas en S3 y la ruta en el equipo local donde se encuentra el archivo de permisos (*.pem*) para la utilización de la cuenta de AWS que se proporciona.

Los campos bajo el nombre de *repo* se corresponden con el nombre completo, la dirección web del repositorio donde se aloja el código fuente de la aplicación se utilizará y la rama sobre la que se desea trabajar

Por último se encuentran los datos para el envío de correos, siendo estos una lista de receptores, el emisor, y el asunto del correo que se enviará, teniendo en cuenta que de momento, esto sólo ocurrirá cuando una nueva AMI sea creada. Todos los correos de estos campos tienen que ser previamente validados desde AWS.

Si los ficheros de configuración no se encuentran en su respectiva carpeta rellenos de una forma correcta, la herramienta no podrá ser utilizada. Del mismo modo, en caso de tener que instalar dependencias, sus ejecutables deberán encontrarse en la carpeta correspondiente.

5.6. Opciones

Una vez conocidos los requisitos de la configuración y las dependencias, pasamos a detallar cada una de las opciones o tareas disponibles en la herramienta, presentes en el fabfile. Se incluye una tabla con las diferentes opciones y sus argumentos en el Anexo A.

Dependencies

La función de instalación de dependencias. Recibe como parámetro el nombre del fichero de dependencias que se quiere instalar (debe ser un script) y éstas se instalarán en cada una de las máquinas asociadas a la aplicación (aquellas especificadas en el archivo de configuración).

En caso de no haber ninguna máquina en la que instalar dependencias, se imprimirá un error que indicará que al menos debe crearse una antes.

Create VM

La función de creación de máquinas virtuales. Tiene como particularidad que en vez de ejecutarse por cada máquina ya creada se hará una única vez para que se cree una sola máquina y no una por cada máquina ya existente.

Toma como parámetro el nombre que se quiera dar a la nueva máquina y una vez termina su creación aparecerán los datos de la misma por pantalla y se actualizará el fichero de configuración añadiendo la máquina a la lista de hosts

Create AMI

La función de creación de AMIs. Al igual que la función de creación de máquinas, esta función se ejecuta una única vez para crear una sola AMI y no una por cada máquina.

Para la creación de una AMI se necesita que se proporcione un identificador de instancia para tener una base sobre la cual crearla, un nombre para nueva AMI y una descripción para la misma. El nombre de las AMIs, según convenio con Telefonica debe estar escrito en mayúsculas y puede contener números pero seguir el siguiente formato: PROJECT-PLATFORM-STATUS, para indicar a qué proyecto, plataformas y estado del proyecto se refiere (entendiendo estado como desarrollo, producción, etc.).

Debido al tiempo que se tarda en la creación de AMIs y a que la información de su creación puede interesar a otras personas involucradas en el mismo proyecto, se ha decidido que cuando el estado de la nueva AMI creada pase a activo se enviará un correo con sus datos a todos los que se encuentren en el fichero de configuración. No se podrá operar sobre esta AMI hasta que no esté disponible, sin embargo, se podrán realizar otras operaciones desde otra consola sin problemas de espera.

Al igual que al crear una nueva máquina, a su término se imprimirán los datos de la nueva AMI por pantalla y se actualizará el identificador en la configuración de la aplicación para usarla como nueva base para las máquinas virtuales.

List AMIs

La función de listado de AMIs. Lista todas las AMIs de la cuenta de AWS que se esté utilizando. En caso de proporcionársele como argumento un proyecto concreto, se listarán únicamente las AMIs de ese proyecto.

SSH

La función de inicio de consola interactiva. Inicia en la máquina local una consola en la máquina remota para trabajar directamente en ella.

Start

La función de arranque de máquinas virtuales. Independientemente del estado en que se encuentren, *start* arranca todas las máquinas virtuales indicadas en el fichero de configuración.

Stop

La función de parada de máquinas virtuales. Independientemente del estado en que se encuentren, *stop* detiene todas las máquinas virtuales indicadas en el fichero de configuración.

Terminate

La función de eliminación de máquinas virtuales. Independientemente del estado en que se encuentren, *terminate* elimina todas las máquinas virtuales indicadas en el fichero de configuración y lo actualiza.

En la interfaz web de AWS las máquinas seguirán apareciendo con estado 'terminate' un periodo de tiempo, sin embargo, ya no se podrá operar en ellas. Pasado este tiempo, desaparecerán.

Init DB

La función de inicio de bases de datos. Por cada máquina indicada en el fichero de configuración, crea una base de datos (por convenio, mysql), un usuario y le da permisos de root.

Create DB

La función sencilla de creación de bases de datos. Dado un nombre y una contraseña para la base de datos facilitados como parámetros, ésta se creará en todas las máquinas virtuales indicadas en el fichero de configuración.

Create user

La función sencilla de creación de usuarios para la base de datos. Dados un nombre de usuario y una contraseña facilitados como parámetro, se creará dicho usuario para el acceso a las bases de datos existentes.

Authorize

La función que otorga permisos de root. Dados como parámetro un nombre de usuario y una contraseña se dan todos los permisos a ese usuario. En caso de proporcionarse además un nombre de base de datos, se darán permisos sólo para aquellas cuyo nombre coincida.

Clone

La función de descarga de código fuente de un repositorio. Se crea una carpeta que tiene como nombre el del repositorio indicado en el archivo de configuración, a continuación se crea e inicia un entorno virtual en el cual se clona el repositorio. Por último se cambia a la rama indicada en la configuración.

La herramienta está diseñada para trabajar con repositorios en Git.

Update

La función de actualización de código. Se realiza un pull de la rama indicada dentro del entorno virtual que se usa para el código fuente, con lo que se actualiza el que tenemos en local.

Help

La función de ayuda. Muestra en pantalla cuáles son y qué hacen cada una de las tareas disponibles en la herramienta.

5.7. Funcionamiento

Para un correcto funcionamiento de la herramienta, debe utilizarse de la siguiente manera:

5.7.1. Instalación

La instalación de la herramienta consiste únicamente en la descompresión de la misma seguida de la instalación de las dependencias de la misma (no confundir con la sección de dependencias de la herramienta, que contiene las de las aplicaciones a utilizar a posteriori) contenidas en el fichero *requirements.txt* utilizando *pip* de la siguiente forma:

```
$> pip install -r requirements.txt
```

Figura 16. Instalación de dependencias de la herramienta

5.7.2. Datos iniciales

Tras la instalación de la herramienta debe procederse a completar todo lo especificado en el apartado *5.5. Requisitos* de este mismo capítulo, poniendo así a punto todos los datos y ficheros iniciales que usaremos con la herramienta.

5.7.3. Ejecución

Con la configuración y las dependencias listas en sus respetivas carpetas y situándonos en la raíz de la herramienta, podemos proceder a ejecutarla mediante el siguiente mandato de consola:

```
$> deptool action [arguments]
```

Figura 17. Mandato de arranque de Deptool

De ese mandato, *action* se corresponde con el nombre de la tarea que queremos ejecutar y *arguments* con cada uno de los parámetros que se corresponden con la tarea seleccionada. Dichas tareas se encuentran especificadas en el apartado anterior de este mismo capítulo e indicadas con sus argumentos en el Anexo A.

5.8. Metodología

En el ciclo de vida del proyecto, durante todas las fases de investigación, diseño y desarrollo de la herramienta se ha seguido la metodología de desarrollo ágil Scrum con *sprints* de dos semanas en las fases de investigación y diseño y de una semana en la fase de desarrollo. La duración de los *sprints* se redujo una vez adquiridos los conocimientos necesarios sobre el Cli y realizados los cambios necesarios en el diseño, ya que se llegó a la conclusión de que las tareas podían avanzar su desarrollo lo suficiente de semana en semana.

5.8.1. Pruebas

En los *sprints* de la fase de desarrollo se pretendía tener al finalizar la semana el número de tareas acordadas completas y probadas. Para este desarrollo se utilizó la técnica denominada TDD (*Test Driven Development*). Para ello, se escribían casos de uso que involucraran las tareas que iban a ser implementadas para después escribir el código que satisficiera dichas pruebas.

Los casos de uso utilizados involucraban, según el caso, dos tipos de pruebas, siendo estas unitarias si demostraban el funcionamiento correcto de una funcionalidad básica o de integración si involucraban la interacción entre diferentes partes de la herramienta.

En este proyecto en algunas de las pruebas se ha hecho uso de entornos reales, ya que no podría comprobarse la funcionalidad sin conectarse a ellos. Siempre que una prueba involucraba cualquier tipo de interacción con un servicio de Amazon, se paraban al finalizar las pruebas todas las máquinas o AMIs iniciadas para evitar consumos innecesarios.

Pruebas unitarias

Las pruebas unitarias se realizaban cada vez que una funcionalidad básica se implementaba por completo, aunque su relevancia o complejidad no fuera elevada para asegurar su correcto funcionamiento y evitar afectar a otras partes del código provocando un error mayor.

Para la elaboración de las pruebas unitarias se ha utilizado la librería de Python unittest [22].

Pruebas de integración

Las pruebas de integración se realizaban al finalizar alguna tarea que involucrara el uso de otras (p. ej. iniciar una base de datos requiere de su creación, de la creación de un usuario y de dar permisos a dicho usuario, tareas más sencillas) o para probar un caso de uso que requiriera varias tareas. Gracias a estas pruebas se puede comprobar la correcta interacción entre las diferentes funcionalidades.

Para la elaboración de las pruebas de integración se ha utilizado la librería de Python lettuce [23].

Prueba del sistema completo

Para finalizar se probó el sistema utilizando únicamente la herramienta y Amazon Web Services, creando varias máquinas, realizando varias operaciones sobre ellas, creando varias AMIs asociadas a varios proyectos y listándolas a continuación, para finalizar parando las máquinas y eliminándolas.

Aunque de forma menos frecuente, el sistema completo se probó más de una vez siguiendo diferentes casos de uso para asegurar el correcto funcionamiento de la herramienta.

5.8.2. Gestión de la configuración

Durante el desarrollo de la herramienta se ha utilizado la herramienta de control de versiones Git siguiendo las recomendaciones marcadas por GitFlow [24] y subiendo todos los cambios realizados a un repositorio privado de Github, usando el sistema de *issues* y *tickets* que este mismo sistema ofrece.

Capítulo 6

Conclusiones y líneas futuras

En este capítulo se presentan las principales conclusiones, técnicas y personales, extraídas de la realización del proyecto, y se perfilan las principales líneas futuras a seguir para su mejora y evolución

6.1. Conclusiones

Si se tiene en cuenta el importante crecimiento que están experimentando en los últimos tiempos los sistemas y aplicaciones distribuidos con la cada vez más necesaria utilización del *Cloud Computing* es inevitable que antes o después tengan que utilizarse herramientas que faciliten su uso. Centrándonos sobre todo en proyectos a gran escala resulta casi imprescindible contar con algo que ayude a desarrolladores y administradores de sistemas a mantener en un estado óptimo toda la infraestructura necesaria para que los proyectos en los que trabajan lleguen a buen término.

Teniendo en cuenta además que todo servicio conlleva un gasto, y sabiendo que el alquiler de hardware es totalmente necesario, conviene abaratar costes creando herramientas como la que se presenta en este proyecto, más sencilla que otras que se encuentran en el mercado pero con la funcionalidad necesaria para no costar más dinero a la empresa con el pago de mensualidades más caras de lo necesario. Aunque con el grado de madurez con el que cuenta Deptool ahora mismo es posible que sea necesario el uso de un cliente sencillo de alguna de estas herramientas para llevar una visión general del proyecto (teniendo siempre en cuenta que se habla de proyectos muy grandes de una empresa internacional), en grupos de trabajo reducidos es perfectamente factible su uso,

pensando en el tedioso trabajo que los administradores de sistemas y desarrolladores tienen que realizar constantemente sin ningún tipo de ayuda.

La automatización de parte de estas tareas no sólo aumenta la productividad en estos grupos de trabajo, sino que reduce los posibles errores u olvidos que podrían cometerse al tener que realizar una misma acción repetidas veces sobre máquinas diferentes, disminuyendo los errores humanos en la medida de lo posible.

Aunque aún quede un largo camino por recorrer para que Deptool sea una herramienta completa, es indudable el potencial con el que cuenta a la hora de realizar proyectos.

Sin ir más lejos, yo misma la he utilizado para proyectos personales con resultados satisfactorios, lo que demuestra que su uso no está restringido únicamente a un tipo de usuarios, sino que está abierto a su utilización de forma más personal, facilitando de igual manera el trabajo a realizar.

A nivel personal, gracias a este proyecto he adquirido muchos conocimientos en el área de las aplicaciones y los sistemas distribuidos, y sobre las ventajas y problemas que su utilización presenta. Además he podido conocer cómo es la administración de sistemas de proyectos en grandes empresas, aprendiendo a gestionar los recursos y bastante a fondo el funcionamiento de muchos de los servicios que se ofrecen en Amazon Web Services, que hasta hace bien poco sólo conocía de oídas. Desarrollando esta herramienta además me ha ayudado a conocer más a fondo las virtudes de Python y sus librerías o a utilizar el control de versiones Git.

Para terminar, el haber trabajado en un entorno profesional para una empresa puntera como es Telefonica I+D me ha proporcionado una gran visión de la forma de trabajo y desarrollo que se lleva en el mundo laboral abriéndome las puertas a lo que puedo encontrar en un futuro y dándome una primera experiencia en este tipo de entornos de trabajo remoto en grupo dependiente de reuniones y requisitos que no había experimentado hasta la fecha.

6.2. Líneas futuras

Conforme avanzaba el proyecto se han ido viendo una serie de ideas que podrían resultar interesante abordar en un futuro para mejorar la experiencia de usuario, así como para ofrecer una funcionalidad más completa.

En primer lugar convendría integrar esta herramienta con la que se ha comentado en capítulos anteriores de control de configuración para asegurar así un manejo más correcto de forma distribuida. En el estado actual de la herramienta, la configuración se maneja únicamente de forma local, por lo que todo cambio no será actualizado para el resto de usuarios, algo inaceptable en aplicaciones distribuidas. En una primera versión del desarrollo se barajó utilizar S3 para subir el archivo de configuración cada vez que se modificara, pero ante la perspectiva de encontrarnos con conflictos y sabiendo que otra herramienta compatible con esta se estaba desarrollando de forma paralela, se desechó la idea pasando a la simpleza del manejo local.

En segundo lugar sería interesante estudiar una forma sencilla de poder realizar operaciones en varias máquinas y no en todas. El problema de que el Cli únicamente realizaba acciones sobre una máquina se ha solventado, y en Deptool se pueden realizar las operaciones sobre y desde todas las máquinas remotas que se encuentren en el fichero de configuración. Sin embargo, no se ha creado un sistema de posibles roles que las agrupen para realizar unas tareas en un único rol. Con esto, por ejemplo, podríamos asignar a un grupo de máquinas el rol de base de datos, evitando así iniciar bases de datos en aquellas que no lo necesiten. Con los decoradores que proporciona Fabric, una vez asignados estos roles sería muy sencillo indicar qué operaciones debería realizar cada rol, e incluso ir más allá seleccionando alguna o algunas máquinas concretas dentro de un mismo rol.

En tercer lugar debería mejorarse el sistema de envío de correos y no utilizarse únicamente para informar de la creación de una nueva AMI. Con la configuración distribuida sería conveniente que cualquier cambio relevante fuera comunicado a los interesados para evitar conflictos. Para alguno de los casos ya podría haberse implementado, como en la eliminación de máquinas, ya que es posible que al no encontrarse la configuración compartida por el momento, otra persona intente acceder a una máquina que ya no existe. Del mismo modo que ahora al cambiar el AMI, dicho campo debería ser cambiado a mano por cada una de las personas afectadas (menos quien ejecutó el mandato desde la

herramienta) en su archivo de configuración, remitiéndonos de nuevo a la primera mejora indicada.

En cuarto lugar podría incorporarse una interfaz gráfica que hiciera más atractiva la herramienta y facilitara su uso para todo tipo de usuarios. Aunque en principio está dirigida a personas familiarizadas con el uso constante de línea de comandos, una interfaz gráfica resultaría más cómoda de utilizar y abriría el abanico de usuarios potenciales. Además, la curva de aprendizaje, aunque en la actualidad no es pronunciada en absoluto, se reduciría aún más si cabe, proporcionando una herramienta práctica y cómoda, más cercana a lo que se ofrece en Amazon Web Services, unificándolos en la medida de lo posible para crear armonía.

Como última propuesta, muchas más acciones podrían ser incorporadas a la herramienta, facilitando aún más el uso de los diferentes usuarios, pero habría que hacerlo de la forma más general posible, no como ya se hacía en la prueba de concepto, donde algunas de las acciones eran demasiado específicas y tuvieron que ser descartadas.

Aunque es probable que de continuar con el desarrollo de Deptool, muchas más propuestas de mejora salgan a relucir, las aquí mencionadas serían más que suficientes para conseguir poco a poco una herramienta más completa, útil y fácil de usar.

Bibliografía

[1] Cisco Systems, Inc. *Infrastructure as a Service: Accelerating Time to Profitable New Revenue Streams. Cisco Infrastructure as a Service*, 2009

[2] Margaret Rouse. *Essential Guide: The AWS re:Invent 2013 experience. Infrastructure as a Service (IaaS),* 2010

[3]Allen B. Downey. *Think Python. How to Think Like a Computer Scientist.* Version 1.3.1. O'Reilly 2012

[4]Arturo Fernández Montoro. *Python al descubierto.* RC Libros, 2012

[5] Mark Lutz. *Learning Python.* Third Edition. O'Reilly, 2008

[6] Brandon Rhodes and John Goerzen. *Foundations of Python Network Programming: The comprehensive guide to building network applications with Python.* Second Edition. Apress, 2010

[7] *Fabric*, http://www.fabfile.org/ , Último acceso en 6 de junio de 2014

[8] Adrian Hannah. *Fabric: a system administrator's best friend.* Linux Journal Volume 2013 Issue 226, Article No. 3, February 2013

[9] *Fabric documentation.* http://docs.fabfile.org/en/1.8/usage/fab.html , Último acceso en 6 de junio de 2014

[10] Jinesh Varia and Sajee Mathew. *Overview of Amazon Web Services*, January 2014

[11] *Boto*, http://boto.readthedocs.org/en/latest/index.html , Último acceso en 6 de junio de 2014

[12] Microsoft Malaysia. MSDN Blogs. *Layered Architecture for .NET.* 2 August 2013 http://blogs.msdn.com/b/malaysia/archive/2013/08/02/layered-architecture-for-net.aspx , Último acceso en 6 de junio de 2014

[13] Douglas E. Comer and David L. Stevens. *Internetworking with TCP/IP Vol III. Client-Server programming and applications.* Second Edition. Prentice Hall, 1993

[14] José Ramón Arias. *Programación de aplicaciones distribuidas usando sockets*. 2ª Versión Oct. 2000

[15] *Chef Documentation*, http://docs.opscode.com/ , Último acceso en 6 de junio de 2014

[16] *Chef Overview*, http://docs.opscode.com/chef_overview.html , Último acceso en 6 de junio de 2014

[17] *Chef*, http://www.getchef.com/chef/ , Último acceso en 6 de junio de 2014

[18] *Puppet*, http://puppetlabs.com/puppet/what-is-puppet , Último acceso en 6 de junio de 2014

[19] *Puppet Documentation*, http://docs.puppetlabs.com/#puppetpuppet , Último acceso en 6 de junio de 2014

[20] *Puppet Enterprise*, http://puppetlabs.com/puppet/puppet-enterprise , Último acceso en 6 de junio de 2014

[21] *Capistrano*, http://capistranorb.com/ , Último acceso en 6 de junio de 2014

[22] *Unittest Documentation*, https://docs.python.org/2/library/unittest.html , Último acceso en 6 de junio de 2014

[23] *Lettuce*, http://lettuce.it/ , Último acceso en 6 de junio de 2014

[24] Git Tutorials. Worlflows Git.
https://www.atlassian.com/es/git/workflows#!workflow-gitflow , Último acceso en 6 de junio de 2014

Anexo A

Tabla resumen de las diferentes tareas (actions) disponibles en la herramienta con sus correspondientes parámetros (arguments).

Actions	Aguments
dependencies	filedep
createVM	name
createAMI	instance_id name description
listAMIS	[project]
ssh	
start	
stop	
terminate	
initDB	
createDb	dbName password
createUser	username password
authorize	username password [dbName]
clone	
update	
help	

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Fri Jun 06 19:50:37 CEST 2014
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)